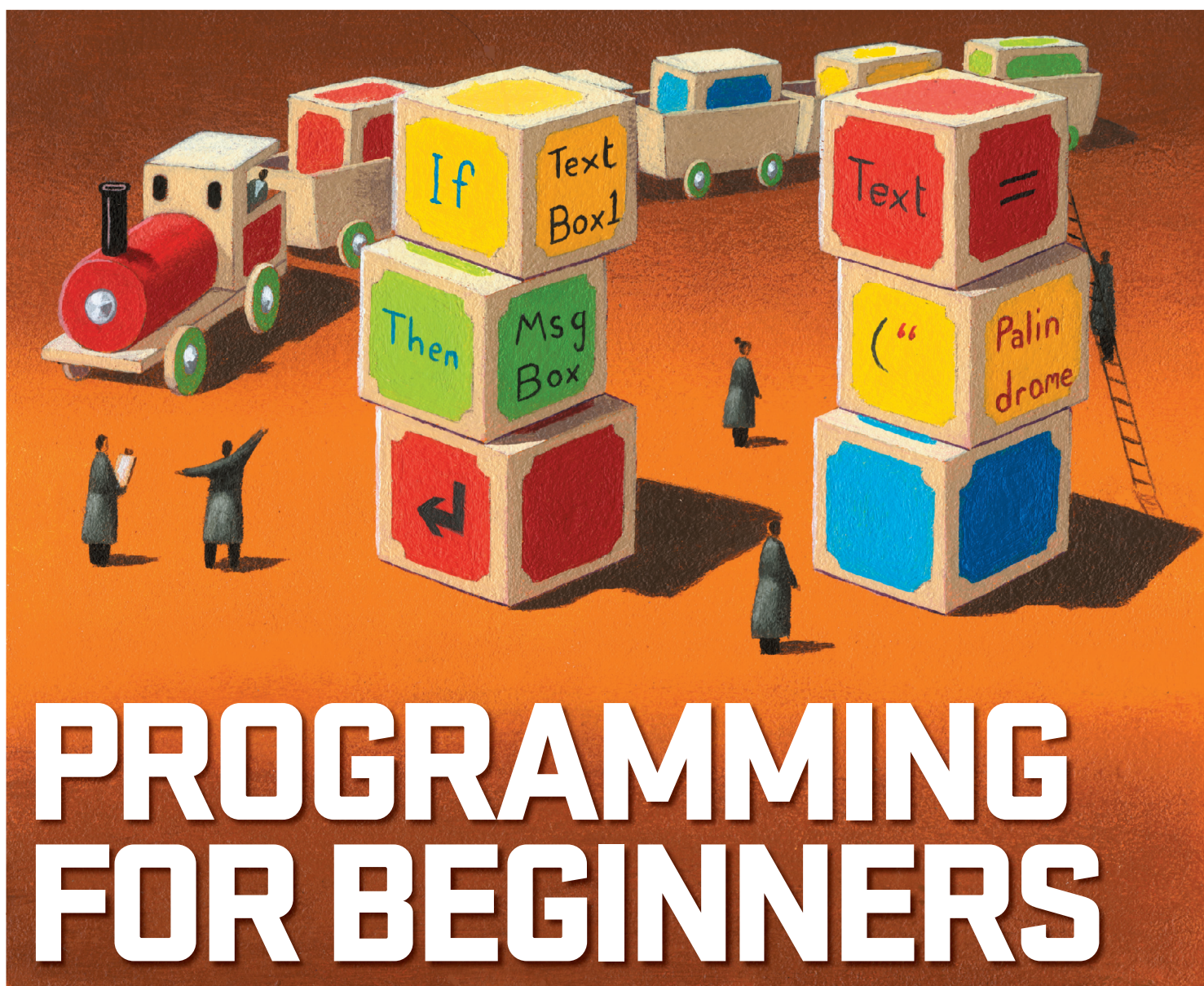




PROGRAMMING FOR BEGINNERS

Programming is a satisfying and potentially rewarding skill, and with Visual Basic 2005 Express Edition, it's surprisingly easy, too. Mike James explains the basics

Programming is fun and is also potentially profitable. From a practical point of view, being able to program means you can make a computer do exactly what you want. You can create your own applications and perhaps even sell them.

An understanding of programming will mean you'll have a better understanding of how software, and hence the computer running it, works, and this helps when troubleshooting. For these practical reasons, learning to program is a good idea. But for many people, the most important reason is simply that it is enjoyable. Programming is an intellectual challenge that goes far beyond Sudoku and is more absorbing and rewarding than many computer games.

Learning to program a PC isn't difficult, especially if you've ever tried programming any other type of computer. But like any skill, it requires some time and effort to master. You can't expect to get it all correct at the first attempt, any more than you would expect to master Sudoku or chess in one sitting.

In this feature, we are going to be using Visual Basic 2005 Express Edition. We've included a CD containing the full version with this month's magazine. This is one of the most productive programming languages and is also powerful enough to create real applications. This article will take you from basic techniques through to the most important ideas of modern programming. You can then hopefully develop these ideas and your skills further.

program, because all it does is display the message 'Hello world'. This simple program has a lot to teach us even though it doesn't appear to be very interesting.

The first time you run VB 2005 Express, it configures itself and presents its startup page, which is full of useful but possibly distracting information, so ignore all this for now. All VB programs are organised into projects. To start a new program, use the menu command File, New Project. This opens the New Project dialog box, which you can use to select the type of

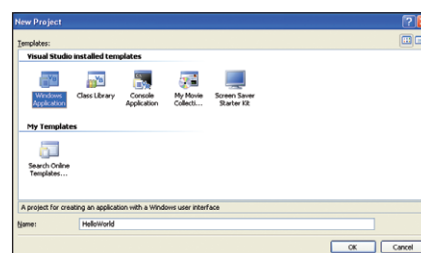
Your bonus cover disc



You'll find a CD containing the full version of Visual Basic 2005 Express Edition on this month's magazine.

BASIC TRAINING

The first program you create with any new language or system should be very simple. It acts as a reality check, and allows you to ensure that you have installed everything correctly and know how to enter and run a program. A suitable program is called the Hello World



▲ Fig 1: Getting started: go to File, New Project and choose Windows Application



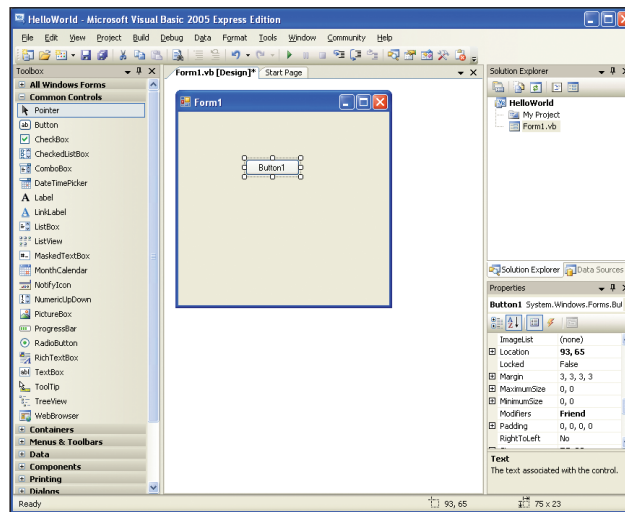
project you want to create and give it a name. Select Windows Application and give it the name HelloWorld (see Fig 1).

A complete project is made up of a number of files and these are created for you automatically. When this is complete, you are presented with the Form designer with the default form, Form1, open and ready for you to work with. Forms are like windows and you use them to create user interfaces. You can customise a form by placing on to it controls such as buttons, menus and other familiar user interface components. Here we want a single button that causes the 'Hello world' message to appear when the user clicks it.

PUSH THE BUTTON

To place a button or any control on a form, open the Toolbox by clicking on its tab on the far left. The button component is in Common Controls. To place it on the form, click the Button component and then click on the form displayed in the designer. This places a default button on the form (Fig 2).

Next, we need to define what will happen when the user clicks the button. This is what programming in more general terms is all about, defining precisely what is going to happen when the program is run. If you double-click on the



◀ Fig 2: To place a button on the form, open the Toolbox and go to Common Controls. Click the Button component and then click the form

button, you are taken from the designer to the code editor. The form designer shows you what the form looks like while the code editor lets you define how it behaves. The code editor already contains some lines of VB ready for you to start work. In this case, the generated code is:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    End Sub
End Class
```

Don't worry about the first and last line; we will return to the Class concept later. The important part is the section from Private Sub to End Sub. These are only two lines, one starting Private Sub and one starting End Sub. Sub is short for subroutine and you can think of it as defining a little program that is part of a larger program. In this case, the 'Handles Button1.Click' part of the definition is most important. This means the subroutine is executed when the user clicks Button1. You need to know some VB commands to put between Sub and End Sub, but this is what learning to program is all about.

GET THE MESSAGE

In this example we will use a simple command that makes a message box appear on the screen. Type the line:

```
MsgBox("Hello VB 2005 World")
```

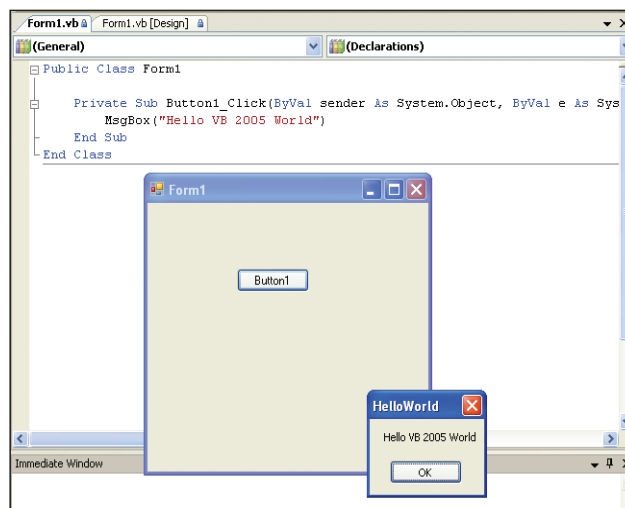
The complete subroutine should be:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    MsgBox("Hello VB 2005 World")
End Sub
```

You have to type it exactly as shown, including the double quotes and make sure it's after the line that starts Private Sub and before End Sub. The editor formats it automatically using colour as it identifies different parts of the command.

To try the program out, click on the green Run arrow icon or use the menu command Debug, Start Debugging. After a pause, the form should appear, complete with button, and when you click on the button the message box should appear (Fig 3). If it doesn't, check that you typed the line exactly as given. If it still doesn't work, at this early stage it's best to start again with a new project. When you first use VB 2005 Express, you can sometimes click something you didn't mean to or that you didn't think was important.

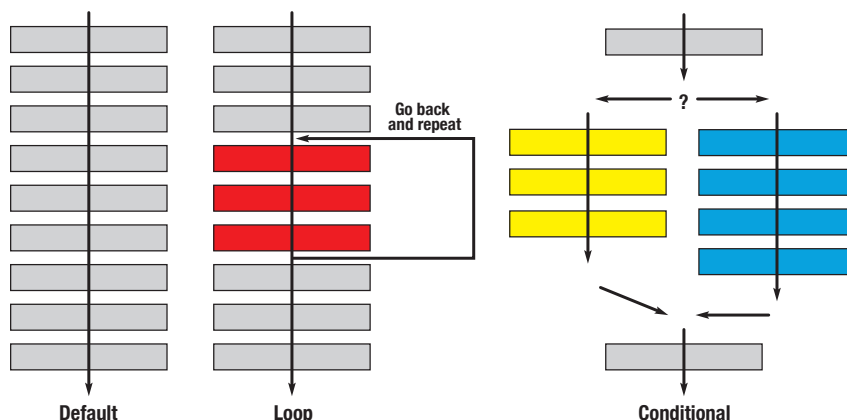
With your first program complete, feel free to play around with what you have created before moving on to the next project. Save the project before continuing.



◀ Fig 3: Clicking on the button should now make the message 'Hello VB 2005 World' appear



Repeating actions and making decisions



The simplest type of program is a list of commands that are obeyed one after the other. But this Default flow is impractical for programs that do the same thing over and over. For this, programmers use a Loop, in which a series of commands are repeated. Intelligent programs need to react to events and make decisions, which requires a Conditional flow, in which different sets of commands are followed in different circumstances.

STAYING IN CONTROL

Initially, most of the programming you do will be like the HelloWorld example. You'll be creating commands that fit between Sub and End Sub statements, which have been automatically generated. You can worry about the generated parts later, as understanding them means grasping the principles of Visual Basic 2005 described collectively as object-oriented programming or OOP; see the box on page 146. For the moment, it's a good idea to focus on the code that makes up subroutines; the fine detail of a program. A theory also underpins what we do here. It's called procedural programming, as we are concerned with what the instructions tell the PC to do.

A program, or more specifically a subroutine, is a list of instructions that the computer obeys at the appropriate time, such as when a button is clicked. A large part of learning VB or any language is discovering what commands are at your disposal. There is also the matter of how you use them to make programs. The most obvious way is just to write one instruction after another:

```
Instruction A
Instruction B
Instruction C
```

Here instructions are obeyed sequentially. This process is called the default flow of control and is one of three general types of flow of control.

The second type is a repeat or loop. It is typified by the English command "Do *something* three times" or "Repeat this exercise until you are too tired". A loop either repeats a given number of times (this is called an enumeration loop) or until some condition is reached (known as a conditional loop). Loops differ greatly in their detail and you'll spend a lot of time mastering them, but keep in mind they are simply a way of repeating instructions. VB has several commands for forming loops and the most important of these are included in our examples.

The third and final type of flow control is the conditional. This is usually expressed as "If *some condition* is true Then do *action*". A more

complicated conditional is "If *something* Then do *this* Else do *that*". This gives a choice between two alternative lists of instructions. Conditionals create branches in the list of instructions with different sections carried out according to the conditions.

You can think of the flow of control through a program as all the possible routes through it. The three fundamental forms are illustrated in the box above. Any program can be built using just these three in combination. This makes practical sense only when you have discovered how to express these ideas in VB, so it's time to move on to the next example.

GOING FURTHER

As you learn more about programming, the examples get more interesting and realistic, but don't run before you can walk. In this example, we'll allow the user to type in a word and check to see if it is a palindrome, that is, it reads the same backwards as forwards. This is most easily done by reversing the letters and seeing if the result is the same as the original word. In this project you will learn how to create a user interface, and how loops and conditionals go together to make programs that do complicated things. We will also construct the program incrementally by adding features and trying things out as we go along.

Start VB 2005 Express and use the command File, New Project. Give the project the name Palindrome. You should be familiar with the default form and designer. The user interface is straightforward. We will provide somewhere for the user to type a word, a button to start the reversal of the letters and, optionally, somewhere to show the reversed word. We also need a way of showing the user if the word is a palindrome or not.

There are a number of controls that can be used to enter and present text to the user. The TextBox control is the simplest. Place two TextBox controls on the form and one Button. These are all found in the Common Controls section of the Toolbox. Alter the size and position them on the form wherever you want them to be (see Fig 4 right). Double-click on the button. You will see the code editor and a

subroutine that handles the button's click event just as in our HelloWorld example. The user enters some text into the first TextBox, which has the name TextBox1 by default. When the user clicks the button, the subroutine must reverse the letters in the word and display them in the second TextBox.

Let's just start by trying to transfer the contents of the first TextBox to the second. To do this, you need to know that a component is an example of an object and that objects have properties and methods. Properties hold values relevant to the object and methods are actions that the object can perform. This will become clear as we develop more programs. What is relevant here is that every TextBox has a Text property that stores what the TextBox displays. With this extra knowledge, we can write the first version of the subroutine:

```
Private Sub Button1_Click(ByVal
    sender As System.Object, ByVal
    e As System.EventArgs) Handles
    Button1.Click
        TextBox2.Text = TextBox1.Text
End Sub
```

The "TextBox2.Text = TextBox1.Text" line copies the content of TextBox1 to TextBox2. The '.' is used as a separator when specifying the object and property you're interested in. An expression such as A.B means the property B belonging to object A. As you type this line in, you will find that VB offers you suggestions about which properties you want to enter after the full stop, and this makes programming much easier.

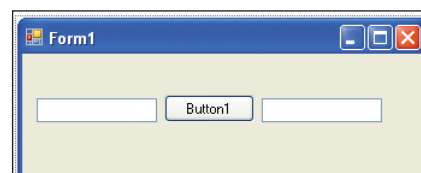
If you run the program as it is, you'll find you can type something into the first box, click the button and it will appear in the second box. Make sure you press the Stop Debugging button when you've finished or you won't be able to edit the next bit of code.

LOOP THE LOOP

So far so good, but to reverse the letters we need to extract each one in turn. Let's write some code that does this. To do this we'll use a type of loop that does the same thing to each item in a set of items. In VB this is called a For Each loop. The instructions in the subroutine have to be changed to read:

```
For Each letter As Char In
    TextBox1.Text
    TextBox2.Text = letter
Next
```

This code is worth examining in detail. For Each tells VB we are going to use a loop to process a set of items. Next comes letter – this is called a variable and it is used to store each item used in the loop. The As Char In TextBox1.Text part tells VB that the items for which you want to repeat the instructions are the characters in the Text property. Everything in VB has an exactly



▲ Fig 4: The Palindrome project needs two TextBox controls on the form and one Button

What is object-oriented programming?

Object-oriented programming (OOP) is the modern way to build software. We build machines from objects, and this is how we should build programs. A software object is modelled on a real-world object. It has properties and methods and can respond to events. By following the projects here, you will get a feel for what properties and events are and see that, in VB, methods are subroutines. But what are the objects they belong to? Look at the code of any of the projects and you'll see such statements as:

```
Public Class Form1
```

This defines a class called Form1. All the lines of code between the Class and the End Class statements constitute a specification for an object, in this case a form. Using the class, you can create as many objects from it as you like. The class-object distinction can be confusing, but if you've done the projects in this article, you have already been working with it. All the components in the Toolbox – Buttons, TextBoxes and so forth – are examples of classes. The Button icon in the Toolbox represents a class (a Button definition) and you can use it to create as

many Button objects as you like. Each is an independent instance of the class, and programmers say they 'instantiate' a class to create an object.

The way classes are used to create any number of objects is a powerful idea, and it allows applications to be built from collections of them, but it isn't the only advantage of OOP. Suppose you need a button that does something a little more than a regular button. In non-OOP languages, you would have to start from scratch and create all the code for a button. Using classes, you don't have to. Instead you create a class using the statement Inherits, as in:

```
Public Class NewButton
    Inherits Button
```

This gives the NewButton class all the behaviour of the Button base class. Now you can customise NewButton by adding and modifying existing behaviours. The beauty of inheritance is that you don't use it often but when you do it saves time.

OOP involves some difficult ideas, but they are worth mastering if you want to become a productive programmer.

defined type. Here we define letter to be a Char type of variable. When you enter the For Each instruction, the code editor helpfully adds the Next for you. Any instructions that you write between For Each and Next are repeated once for each of the characters stored in TextBox1.Text. In this case we transfer each letter into TextBox2.Text.

If you now try the program out, you might think something isn't working. No matter what you store in TextBox1, you only get the last letter in TextBox2 when you click the button. This isn't a bug; it's a misunderstanding. Each time a letter is stored in TextBox2.Text it overwrites what is already stored there. The letters go into TextBox2 one by one, but so quickly you don't see them go by; you see only the last one. To see the entire word, we need to add each letter to what is already stored in TextBox2. To do this, you need to know that '&' will add (concatenate) one string of characters to another.

Say you change the line within the For Each loop to read:

```
TextBox2.Text = TextBox2.Text & letter
```

If you do this, you will see the whole word in TextBox1 appear in TextBox2. You might think this is where we started. The first time the loop is obeyed, the first letter in the word ABCD is added to TextBox2.Text to give A. The second time the loop is obeyed, the second letter (B) is added to TextBox2.Text and so on round the loop until we have ABCD and the loop ends.

How can we change this to reverse the characters? Consider what happens if we form:

```
TextBox2.Text = letter &
    TextBox2.Text
```

This adds the letter to the end of what is in TextBox2.Text, whereas the original version

added it to the end. Now the first time through the loop, we have A as before, but the second time through the loop we have B & A, which gives BA. The third time through the loop adds a C to the start, and so on, until we have DCBA. In programming, seemingly small and subtle changes can often make huge differences to what a program does.

If you make the change and run the program, you will see that it does reverse any word you care to enter into TextBox1. However, if you try entering a second word and clicking the button again, you will discover a real bug. The previous word isn't removed from TextBox2 before the new word is added. The solution is to add this line:

```
TextBox2.Text = ""
```

Add it before the For Each loop. This sets TextBox2 to the special text called the null string, the text string with no characters in it. Enter this using double quotes with nothing between, not even a space character.

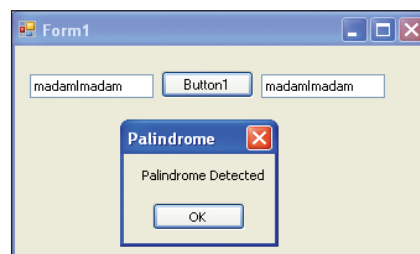
The final refinement is to signal that a palindrome has been detected to the user. The condition for this to be the case is simply:

```
TextBox1.Text = TextBox2.Text
```

The VB command for conditional execution is similar to the equivalent English, and you should be able to understand the following addition to the subroutine:

```
If TextBox1.Text = TextBox2.Text Then
    MsgBox("Palindrome Detected")
End If
```

If you type this at the end of the subroutine, you'll see that the code editor generates the End If automatically for you. All the instructions



▲ Fig 5: The program will now alert the user if a palindrome is detected

between the Then and End If are obeyed if the condition specified between the If and the Then is true. If the condition isn't true, the instructions are skipped and execution continues with whatever follows End If. So if the contents of the two TextBoxes match exactly, a message box pops up and announces the fact. If you want to try it out, the entire final subroutine is:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    TextBox2.Text = ""
    For Each letter As Char In TextBox1.Text
        TextBox2.Text = letter &
            TextBox2.Text
    Next
    If TextBox1.Text = TextBox2.Text Then
        MsgBox("Palindrome Detected")
    End If
End Sub
```

If you enter madamImadam you will trigger the message box (Fig 5), but if you enter MadamImadam, you won't. Sadly we don't have space here to delve into the ways in which the program treats lower and upper case. If you want to know more, look up LCase and UCase in the Help Files of Visual Basic 2005 Express.

WALK THE LINE

Our final program involves graphics. This program is a big jump in both the number of ideas and the techniques it uses. But it's good to see how graphics work in programming as soon as possible. In this project, we'll create a screensaver-type display suitable for keeping users amused. Given the simplicity of the technique, the results are amazing. The idea is simply to repeat drawing a line from one corner of a rectangle to the other, slowly rotating the line and changing its colour.

Start Visual Basic 2005 Express, create a new Windows application project and call it CrazyLine. Use the Toolbox to place on it a single button, a PictureBox and a Timer. You will find the Timer under Components. This is an example of a component that doesn't have a visual presence when you run the program. As a result of this, when you place it on the form it is automatically shifted to a special area in the designer that is reserved for non-visual components.

So far, we have allowed components to retain their default properties to make things simple. Now it's time to meet the Property Window. If you select the Button in the form designer, you'll see that the panel to the bottom



right changes to display all its properties. You can use the Property Window to examine or change any property. Scroll down until you find Button1's Text property and change it from Button1 to Start Graphics. This is then displayed as the button's label (Fig 6 on the right).

We'll start building the program by drawing a single line across the PictureBox. A key idea in VB 2005 is that there is a graphics object, which has methods that allow you to draw on its display surface. Eventually we want to draw lots of lines. This might make you think we need a loop, but as we intend to draw a line every tenth of a second, say, a better approach is to use a Timer control. This will trigger a regular event and we'll do the drawing within the subroutine that handles that event.

To trigger the drawing process, we just set the Timer's interval property to a delay in thousandths of a second and then set its enable property to True. This starts the Timer generating tick events. Double-click the Button in the form designer and enter the following into the Button Click event handler:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Timer1.Interval = 100
    Timer1.Enabled = True
End Sub
```

If you return to the form designer and double-click on the Timer, you will be transferred back to the code editor with a generated Timer Tick event handler. This is where all the drawing is performed. Positions within the PictureBox are given as a number of pixels horizontally (the x-coordinate) and number of pixels down (the y-coordinate). So the top left-hand corner is x=0, y=0 and the bottom right-hand corner is x=width of box, y=height of box. The instructions to draw a single line from the top left-hand corner to the bottom right are:

```
Dim g As Graphics
g = PictureBox1.CreateGraphics()
g.DrawLine(Pens.Black, 0, 0, PictureBox1.Width - 1, PictureBox1.Height - 1)
```

The first line creates a name (g) for the Graphics object that we are about to create. The second line gets the actual Graphics object that represents the drawing surface of the

PictureBox. Once we have this in g, we can use its drawing methods – in this case, DrawLine. If you run this program, clicking on the Start Graphics button will draw a black diagonal line across the PictureBox. Now we just have to move the locations of the start and end of the line each time the timer ticks.

Our next step is to make the line move by rotating it through an angle at each tick. Instead of rotating the line, we can rotate the coordinate system used by the PictureBox and then draw the same line. You can think of this as being analogous to putting a ruler on a piece of paper and rotating the ruler or rotating the paper underneath it each time you draw a line.

The graphics object always performs rotations about the origin – that is, the 0,0 point of the coordinate system, which is at the top left-hand corner of the drawing surface by default. To make the rotation happen around the middle, we have to move the origin of the coordinate system to the middle of the PictureBox. This is easy once you've discovered the Graphic object's TranslateTransform and RotateTransform methods.

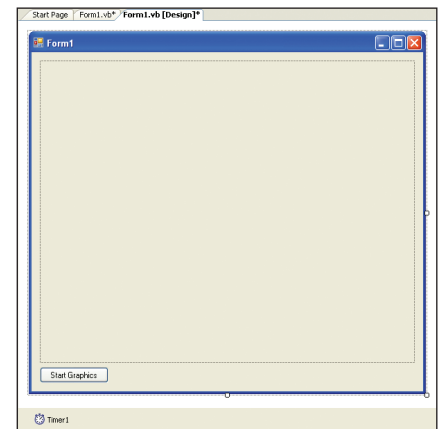
COUNTING TICKS

We're almost ready to write the new version of the Tick event handler, but we need one more idea. Normally when a subroutine is called, it starts running as if it was the first time it existed; it has no memory of any previous uses. We would like to keep count of how many times it has been called – in other words, the total number of ticks. The reason is that we are going to use this number to provide the angle and a colour change to the line as it rotates. We need to declare explicitly a Static variable, one that isn't destroyed and re-created each time the subroutine is called. A Static variable thus provides a way for a subroutine to have a memory of previous use.

The new version of the Timer Tick event starts off declaring the static counting variable:

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick
    Static count As Integer
```

This creates count just once, the first time that the subroutine is used, and sets its value equal to zero. As Integer specifies the type of data we can store in count. Here this is integers (whole



▲ Fig 6: The Button's Text property allows you to change the words written on it

numbers) between -2,147,483,648 and 2,147,483,647.

Each time the subroutine is called (that is, at each tick), we want to add one to count, which is written in VB as:

```
count = count + 1
```

We need the width and height properties of the PictureBox repeatedly so it makes sense to get them once and store them for repeated use:

```
Dim w As Integer
Dim h As Integer
w = PictureBox1.Width
h = PictureBox1.Height
```

Again, the variables are declared to be Integers. Next, we get the graphics object that represents the drawing surface of the PictureBox:

```
Dim g As Graphics
g = PictureBox1.CreateGraphics()
```

We move the origin of the coordinate system to the middle of the PictureBox and rotate by count degrees:

```
g.TranslateTransform(w / 2, h / 2)
g.RotateTransform(CSng(count))
```

The only complication that we have here is that count is an integer and RotateTransform needs an angle specified as a single precision number. CSng converts a numeric value to single precision.

What is .NET?

Visual Basic 2005 Express Edition is based on 'NET' technology. Microsoft made a huge investment in .NET when it was launched in 2002. Back then the entire Windows system was beginning to look old and poorly designed. The .NET system was designed to sweep away the bug-ridden and overly complex ways of doing things. It was to supersede complex or old-fashioned technologies such as COM, ActiveX and most programming APIs that had accumulated in Windows over the years. It hasn't achieved this goal yet, but it is moving closer to the ideal with each new version.

.NET is composed of three major components. There is the Common Language Runtime (CLR), which supports the basic requirements of any .NET language. The CLR is a standard environment that any .NET language can expect to find. Then there is the Framework, a large collection of objects that replace parts of, and add to, the Windows API. The Framework is a huge and expandable kit of parts that you can use to build applications. Finally, there are the .NET languages, which run under the CLR and use the Framework. There are lots of .NET languages, but Microsoft produces and

supports four as part of Visual Studio: C#, Visual Basic, J# (a Java-like language) and Managed C++. Much of the work involved in creating a .NET application is using the Framework components, which are the same in any language, so it can be argued that distinctions between languages are becoming smaller.

Although .NET is a Windows system, there are projects under way to create versions of the entire system that work under other operating systems. The most likely to succeed is Mono. For more details, visit www.mono-project.com.



All that remains now is to draw the line. But to make the display more interesting, we generate a pen in a colour that depends on the rotation angle:

```
Dim p As New Pen(Color.FromArgb
((count + 157) * 3 Mod 255, (count
+ 356) * 7 Mod 255, (count + 873) *
5 Mod 255))
```

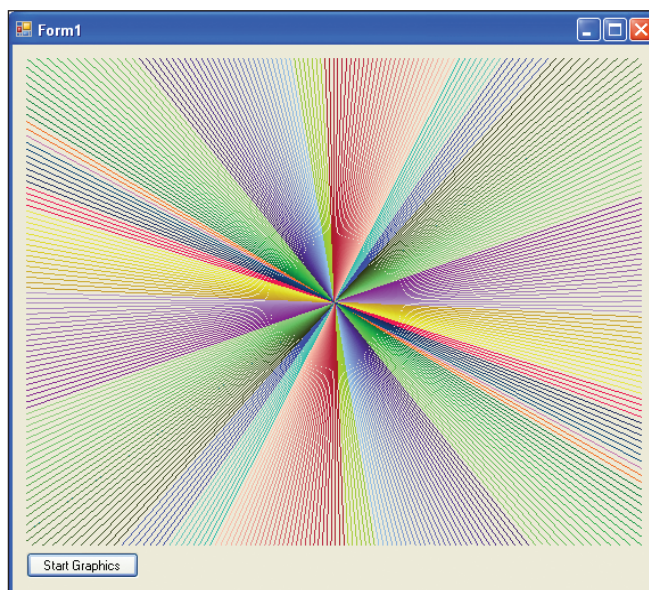
This looks complicated, but the FromArgb function simply creates a colour from a specified amount of red, green and blue. As each of these has to be in the range 0 to 255, the Mod function is used to keep the computed value in this range. The form of the arithmetic and the numbers used were invented by trial and error and what looked good, so feel free to modify the maths.

Finally, we draw the line using the pen to specify the colour:

```
g.DrawLine(p, -w \ 2, -h \ 2, w \ 2,
h \ 2)
```

If you put all this together and run the program, you should see an interesting display (Fig 7). It's easy to create even more interesting displays with just small changes to what is drawn.

It is worth mentioning that there is a bug in this program that becomes apparent only if you leave it running for a long time. Think what



◀ Fig 7: The CrazyLine program creates a screensaver-type display that uses colour to make the image more interesting as it grows

happens when count reaches the largest value it can store.

WHAT NEXT?

This article has been a brief introduction to VB programming. To go further you have to carry on programming. There are lots of online resources that can help and the links in the Help

menu are a good place to start. You can also find a course on VB at <http://visualbasic.about.com>.

There are many books on VB .NET. We reviewed three good ones in the box below. When you feel ready for something more serious, think up a project and go for it. You'll find you could have written it better by the time you have finished it, but that's programming. **CS**

Recommended books Improve your programming skills

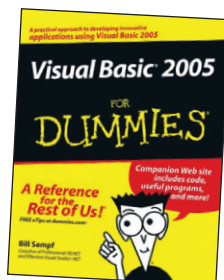
HUNGRY MINDS INC Visual Basic 2005 for Dummies

★★★★★

£17

Bill Sempfl introduces programming in general and Visual Basic 2005 at a slow and steady pace. It's very well written and lacks the corny humour found in other Dummies titles. The only negative point is that it covers the full Visual Studio version of the product; while there is a Dummies book specifically aimed at the Express version, it isn't as good.

If you are having trouble understanding what programming is all about, this book is highly recommended. It shouldn't take you long to outgrow it, but that's the mark of a good introduction.



SUMMARY

- ✓ Easy-to-read introduction
- ✗ Based on full Visual Studio

SUPPLIER www.amazon.co.uk
ISBN 076457728X
PAGES 362

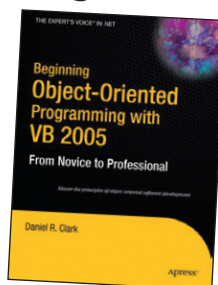
APRESS Beginning Object-Oriented Programming with VB 2005

★★★★★

£22

For an insight into the background of object-oriented programming, the theory and how it applies to VB in particular, Dan Clark's book should be high on your list. It covers important topics such as UML modelling and moves on to how object-oriented ideas are built into VB 2005. It uses lots of short and easy-to-understand examples to make it clear how theory becomes practice. The final part deals with topics such as business logic and working with a database.

To progress from amateur to professional programmer, this book is a good place to start.



SUMMARY

- ✓ Good introduction to the subject
- ✗ Not a fun book

SUPPLIER www.bookfellas.co.uk
ISBN 1590595769
PAGES 384

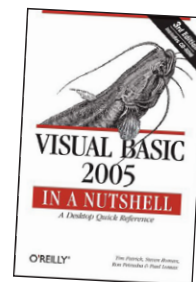
O'REILLY Visual Basic 2005 in a Nutshell

★★★★★

£35.50

If you get on with VB's online documentation, you may not need a reference book. But if you prefer to flick through a book, this one, written by Tim Patrick, Steven Roman and Ron Petruscha, is ideal. It's not the sort of book you sit down and read from cover to cover, but you are likely to reach for it when you come across a problem or a gap in your understanding.

The first part of the book consists of short introductions to various general topics. But the largest part is a reference guide to the VB functions, statements and objects you'll be using to build programs. The result is a genuinely readable reference.



SUMMARY

- ✓ Readable reference on VB 2005
- ✗ Online documentation is good, too

SUPPLIER www.amazon.co.uk
ISBN 059610152X
PAGES 766